Amendments to the claims (this listing replaces all prior versions):

1. (Currently Amended) A computer program product for representing and implementing a concept ~~with~~ from a first functional domain ~~and~~ within a second functional domain, wherein the first functional domain includes a first component representing the concept in ~~a~~ the first functional domain, the first component having a type and a first semantic usability in the first functional domain, and wherein the computer program product comprises:

a computer-readable medium having computer-readable signals stored thereon,

wherein the signals define a proxy component representing the concept in the second functional domain, the proxy component defined to cause execution of the first component in the first functional domain, and

wherein the proxy component has a second semantic usability in the second functional domain and the second semantic usability closely corresponds to the first semantic usability,

the proxy having been generated by analyzing the first component to determine its type.

2. (Original) The computer program product of claim 1, wherein the proxy component was generated by a transformation performed on the first component.

3. (Original) The computer program product of claim 1, wherein the first semantic usability of the first component is defined at least by a first mutability, and

wherein the second semantic usability of the proxy component is defined at least by a second mutability of the proxy component determined from at least the first mutability of the first component.

4. (Original) The computer program product of claim 1, wherein the first semantic usability of the first component is defined at least by a first accessibility, and

wherein the second semantic usability of the proxy component is defined at least by a second accessibility of the proxy component determined from at least the first accessibility of the first component.

5. (Original) The computer program product of claim 1, wherein the first semantic usability of the first component is defined at least by a first instantiability, and

wherein the second semantic usability of the proxy component is defined at least by a second instantiability of the proxy component determined from at least the first instantiability of the first component.

6. (Original) The computer program product of claim 1, wherein the first semantic usability of the first component is defined at least by a first inheritability, and

wherein the second semantic usability of the proxy component is defined at least by a second inheritability of the proxy component determined from at least the first inheritability of the first component.

7. (Original) The computer program product of claim 1, wherein the first semantic usability of the first component is defined at least by a first context usability, and

wherein the second semantic usability of the proxy component is defined at least by a context usability of the proxy component determined from at least the first context usability of the first component.

8. (Original) The computer program product of claim 1, wherein the first semantic usability of the first component is defined at least by a first polymorphicability, and

wherein the second semantic usability of the proxy component is defined at least by a polymorphicability of the proxy component determined from at least the first polymorphicability of the first component.

9. (Original) The computer program product of claim 1, wherein the first functional domain is the Java programming language and the second functional domain is the C++ programming language.

10. (Original) The computer program product of claim 9, wherein the first component is a first Java component and the proxy component is a C++ proxy component, and

wherein the C++ proxy component is aware of a context in which it is defined.

11. (Original) The computer program product of claim 9, wherein the first component is a first Java component and the proxy component is a C++ proxy component, and

wherein the C++ proxy component includes one or more proxy support elements.

12. (Original) The computer program product of claim 11, wherein the C++ proxy component is a C++ proxy class.

13. (Original) The computer program product of claim 12, wherein one or more proxy support elements of the C++ proxy class allow an instance of the C++ proxy class to be context-aware.

14. (Original) The computer program product of claim 13, wherein one or more of the proxy support elements that allow an instance of the C++ proxy class to be context-aware are context constructors.

15. (Original) The computer program product of claim 13, wherein one of the proxy support elements allows an instance of the C++ proxy class to be aware of being a C++ proxy instance field.

16. (Original) The computer program product of claim 13, wherein one of the proxy support elements allows an instance of the C++ proxy class to be aware of being a C++ proxy static field.

17. (Original) The computer program product of claim 13, wherein one of the proxy support elements allows an instance of the C++ proxy class to be aware of being a C++ proxy array element.

18. (Original) The computer program product of claim 13, wherein one of the proxy support elements allows an instance of the C++ proxy class to be aware of being a C++ proxy stand-alone object.

19. (Original) The computer program product of claim 13, wherein the C++ proxy class includes a proxy layer, and awareness of the contexts is required by the proxy layer.

20. (Original) The computer program product of claim 19, wherein the proxy layer is coded using the Java Native Interface.

21. (Original) The computer program product of claim 9, wherein the first component is a Java package and the proxy component is a C++ namespace.

22. (Original) The computer program product of claim 9, wherein the first component is a Java class and the proxy component is either a C++ proxy class or a C++ proxy struct.

23. (Original) The computer program product of claim 22, wherein the proxy component includes one or more proxy support elements.

24. (Original) The computer program product of claim 23, wherein one or more of the proxy support elements allow an instance of the C++ proxy class to be context-aware.

25. (Original) The computer program product of claim 22, wherein the Java class is declared abstract, and the C++ proxy component is instantiable.

26. (Original) The computer program product of claim 9, wherein the first component is a Java array type and the proxy component is either a C++ proxy class or a C++ proxy struct.

27. (Original) The computer program product of claim 26, wherein the proxy component includes one or more proxy support elements.

28. (Original) The computer program product of claim 27, wherein one or more of the proxy support elements allow an instance of the proxy component to be context-aware.

29. (Original) The computer program product of claim 26, wherein the proxy component includes a length field corresponding to a length attribute of the Java array.

30. (Original) The computer program product of claim 26, wherein the Java array has an element type and the proxy component has an element type corresponding to the Java array's element type, and wherein the proxy component includes one or more array subscript operators that return a context-aware instance of the proxy component's type.

31. (Original) The computer program product of claim 26, wherein the Java array has a primitive element type, and the proxy component has a primitive element type corresponding

to the Java array's element type and includes one or more array subscript operators that return a primitive value.

32. (Original) The computer program product of claim 9, wherein the first component is a Java interface and the proxy component is either a C++ proxy class or a C++ proxy struct.

33. (Original) The computer program product of claim 32, wherein the first component includes one or more proxy support elements.

34. (Original) The computer program product of claim 33, wherein one or more of the proxy support elements allow an instance of the proxy component to be context-aware.

35. (Original) The computer program product of claim 32, wherein the proxy component is instantiable.

36. (Original) The computer program product of claim 9, wherein the first component is a Java method and the proxy component is a C++ proxy method.

37. (Original) The computer program product of claim 36, wherein the C++ proxy method has a constness determined from a mutability of the Java method.

38. (Original) The computer program product of claim 36, wherein the C++ proxy method declares a return type that has a mutability determined from a mutability of a return type declared by the Java method.

39. (Original) The computer program product of claim 36, wherein the C++ proxy method passes one or more arguments, each argument having a mutability determined from a mutability of a corresponding argument passed by the Java method.

40. (Original) The computer program product of claim 36, wherein the C++ proxy method is defined to throw an exception based on the Java method being defined to throw an exception.

41. (Original) The computer program product of claim 36, wherein the C++ proxy method is defined not to throw an exception based on the Java method being defined to throw an exception.

42. (Original) The computer program product of claim 36, wherein whether the C++ proxy method is declared virtual or not declared virtual is determined from one or more of the following aspects of the Java method: polymorphicability, mutability, and instantiability.

43. (Original) The computer program product of claim 9, wherein the first component is a Java field and the proxy component is a C++ proxy field.

44. (Original) The computer program product of claim 43, wherein the C++ proxy field is declared to be of type C++ proxy class, the C++ proxy class including one or more proxy support elements that allow an instance of the C++ proxy class to be context-aware, such that an instance of the C++ proxy field is context-aware.

45. (Original) The computer program product of claim 43, wherein the Java field is of primitive type and the C++ proxy field is declared to be of type C++ primitive proxy class.

46. (Original) The computer program product of claim 45, wherein the C++ primitive proxy class includes one or more proxy support elements that allow an instance of the C++

primitive proxy class to be context-aware, such that an instance of the C++ proxy field is context-aware.

47. (Original) The computer program product of claim 45, wherein one or more proxy support elements of the C++ primitive proxy class is an overloaded operator that permits instances the C++ primitive proxy class 52 to be used on the left-hand side of one or more syntactical productions.

48. (Original) The computer program product of claim 43, wherein the C++ proxy field has a mutability determined from a mutability of the Java field.

49-75. (Cancelled)

76. (Original) A system for modeling a first component of a first functional domain, wherein the first component defines a first concept and includes one or more subcomponents, the system comprising:

means for receiving the first component; and

means for generating a first model of the first component, including:

means for generating, for each subcomponent of the first component, a discrete element of the first model to represent the subcomponent; and

means for providing the first model with a property of relationship awareness such that, if a first discrete element or attribute of the first model is changed, the first model is operative to:

determine if the first discrete element or attribute has one or more elements and attributes related to the first discrete element or attribute,

if the first discrete element or attribute has one or more related elements and attributes, determine whether to change the one or more related elements, and

if it is determined to change one or more related elements and attributes, change such one or more elements and attributes in accordance with the changed first discrete element or attribute.

77. (Original) A system for modeling a first component of a first functional domain, wherein the first component defines a first concept and includes one or more subcomponents, the system comprising:

a model generator to receive the first component, to generate a first model including:

for each subcomponent of the first component, a discrete element of the first model representing the subcomponent; and

one or more model concepts that provide the first model with a property of relationship awareness such that, if a first discrete element or attribute of the first model is changed, the first model is operative to:

determine if the first discrete element or attribute has one or more elements and attributes related to the first discrete element or attribute,

if the first discrete element or attribute has one or more related elements and attributes, determine whether to change the one or more related elements, and

if it is determined to change one or more related elements and attributes, change such one or more elements and attributes in accordance with the changed first discrete element or attribute.

78. (Original) A computer program product for modeling a first component of a first functional domain, wherein the first component defines a first concept and includes one or more subcomponents, the system comprising:

a computer-readable medium having computer-readable signals stored thereon,

wherein the signals define a model generator to receive the first component, to generate a first model including:

for each subcomponent of the first component, a discrete element of the first model representing the subcomponent; and

one or more model concepts that provide the first model with a property of relationship awareness such that, if a first discrete element or attribute of the first model is changed, the first model is operative to:

determine if the first discrete element or attribute has one or more elements and attributes related to the first discrete element or attribute,

if the first discrete element or attribute has one or more related elements and attributes, determine whether to change the one or more related elements, and

if it is determined to change one or more related elements and attributes, change such one or more elements and attributes in accordance with the changed first discrete element or attribute.

79. (Currently Amended) A method of transforming a first component of a first domain to a proxy component of a second domain, wherein the first component has a type and defines a first concept, the method comprising acts of:

(a) analyzing ~~determining a type of~~ the first component to determine its type; and

(b) transforming the first component into the proxy component in accordance with the determined type, wherein the proxy component defines at least the concept defined by the first component.

80. (Original) The method of claim 79, wherein the first component is in parsed form.

81. (Original) The method of claim 79, wherein the first component is a source component.

82. (Original) The method of claim 79, wherein the first component is a compiled component.

83. (Original) The method of claim 79, wherein act (b) includes:

(i) generating a model from the first component; and

(ii) generating the proxy component from the model.

84. (Original) The method of claim 79, wherein the proxy component has a semantic usability in the second domain closely corresponding to the semantic usability of the first component in the first domain.

85. (Original) The method of claim 84, wherein the proxy component includes one or more proxy support elements to allow an instance of the proxy component to be context-aware.

86. (Original) The method of claim 79, wherein the first domain is a first programming language.

87. (Original) The method of claim 86, wherein the first programming language is a high-level programming language.

88. (Original) The method of claim 87, wherein the first programming language is Java.

89. (Original) The method of claim 88, wherein the second domain is C++.

90. (Original) The method of claim 89, wherein:
act (a) includes determining that the type of the first component is a Java class, and
act (b) includes transforming the Java class into a C++ proxy class.

91. (Original) The method of claim 90, wherein act (b) includes:

generating, within the C++ proxy class, one or more proxy support elements.

92. (Original) The method of claim 89, wherein:
act (a) includes determining that the type of the first component is a Java array, and
act (b) includes transforming the Java array into a C++ proxy class.

93. (Original) The method of claim 92, wherein act (b) includes:
generating, within the C++ proxy class, one or more proxy support elements.

94. (Original) The method of claim 89, wherein:
act (a) includes determining that the type of the first component is a Java method, and
act (b) includes transforming the Java method into a C++ proxy method.

95. (Original) The method of claim 89, wherein:
act (a) includes determining that the type of the first component is a Java field, and
act (b) includes transforming the Java field into a C++ proxy field.

96. (Original) The method of claim 79, wherein the first semantic usability of the first component is defined at least by a first mutability, and act (b) includes:

(i) determining a second mutability of the proxy component from at least the first mutability of the first component, wherein the second semantic usability of the proxy component is defined at least by the second mutability.

97. (Original) The method of claim 79, wherein the first semantic usability of the first component is defined at least by a first accessibility, and act (b) includes:

(i) determining a second accessibility of the proxy component from at least the first accessibility of the first component, wherein the second semantic usability of the proxy component is defined at least by the second accessibility.

98. (Original) The method of claim 79, wherein the first semantic usability of the first component is defined at least by a first instantiability, and act (b) includes:

(i) determining a second instantiability of the proxy component from at least the first instantiability of the first component, wherein the second semantic usability of the proxy component is defined at least by the second instantiability.

99. (Original) The method of claim 79, wherein the first semantic usability of the first component is defined at least by a first inheritability, and act (b) includes:

(i) determining a second inheritability of the proxy component from at least the first inheritability of the first component, wherein the second semantic usability of the proxy component is defined at least by the second instantiability.

100. (Original) The method of claim 79, wherein the first semantic usability of the first component is defined at least by a first context usability, and act (b) includes:

(i) determining a second context usability of the proxy component from at least the first context usability of the first component, wherein the second semantic usability of the proxy component is defined at least by the second context usability.

101. (Original) The method of claim 79, wherein the first semantic usability of the first component is defined at least by a first polymorphicability, and act (b) includes:

(i) determining a second polymorphicability of the proxy component from at least the first polymorphicability of the first component, wherein the second semantic usability of the proxy component is defined at least by the second polymorphicability.

102. (Currently Amended) A system for transforming a first component of a first domain to a proxy component of a second domain, wherein the first component has a type and defines a first concept, the system comprising:

means for analyzing ~~determining a type of~~ the first component to determine its type; and

means for transforming the first component into the proxy component in accordance with the determined type, wherein the proxy component defines at least the concept defined by the first component.

103. (Currently Amended) A system for transforming a first component of a first domain to a proxy component of a second domain, wherein the first component has a type and defines a first concept, the system comprising:

a component transformer to receive as input the first component, to analyze ~~determine a type of~~ the first component to determine its type, to transform the first component into the proxy component in accordance with the determined type, and to output the proxy component,

wherein the proxy component defines at least the concept defined by the first component.

104. (Currently Amended) A computer program product for transforming a first component of a first domain to a proxy component of a second domain, wherein the first component has a type and defines a first concept, the computer program product comprising:

a computer-readable medium having computer-readable signals thereon,

wherein the signals define a component transformer to receive as input the first component, to analyze ~~determine a type of~~ the first component to determine its type, and to transform the first component into the proxy component in accordance with the determined type,

wherein the proxy component defines at least the concept defined by the first component.

105. (Original) The computer program product of claim 12, wherein at least one of the proxy support elements of the C++ proxy class allows usage of null in C++ corresponding to usage of a null Java object reference in Java.

106. (Original) The computer program product of claim 105, wherein the C++ proxy class includes a conversion constructor to create a stand-alone proxy instance of the C++ proxy class initialized to not refer to any Java instance.

107. (Original) The computer program product of claim 12, wherein at least one of the proxy support elements provides a semantic usability to the C++ proxy class that closely corresponds to the semantic usability of a Java cast expression corresponding to the Java component.

108. (Original) The computer program product of claim 107, wherein the C++ proxy class includes a static class method that provides the semantic usability to the C++ proxy class that closely corresponds to the semantic usability of a Java cast expression corresponding to the Java component.

109. (Original) The computer program product of claim 12, wherein at least one of the proxy support elements provides an ability to map an instance of the C++ proxy class to an object that represents the Java component.

110. (Original) The computer program product of claim 109, wherein the C++ proxy class includes a framework support method that provides the ability to map an instance of the C++ proxy class to an object that represents the Java component.

111. (Original) The method of claim 89, wherein act (b) includes:

(i) transforming the Java component into a C++ proxy class that includes one or more proxy support elements.

112. (Original) The method of claim 111, wherein one or more proxy support elements of the C++ proxy class allow an instance of the C++ proxy class to be context-aware.

113. (Original) The method of claim 112, wherein one of the proxy support elements allows an instance of the C++ proxy class to be aware of being a C++ proxy instance field.

114. (Original) The method of claim 112, wherein one of the proxy support elements allows an instance of the C++ proxy class to be aware of being a C++ proxy static field.

115. (Original) The method of claim 112, wherein one of the proxy support elements allows an instance of the C++ proxy class to be aware of being a C++ proxy array element.

116. (Original) The method of claim 112, wherein one of the proxy support elements allows an instance of the C++ proxy class to be aware of being a C++ proxy stand-alone object.

117. (Original) The method of claim 112, wherein act (b)(i) includes:

generating a proxy layer, and wherein awareness of the contexts is required by the proxy layer.

118. (Original) The method of claim 117, wherein act (b)(i) includes:

coding the proxy layer using the Java Native Interface.

119. (Original) The method of claim 111, wherein at least one of the proxy support elements of the C++ proxy class allows usage of null in C++ corresponding to usage of a null Java object reference in Java.

120. (Original) The method of claim 119, wherein act (b)(i) includes:

generating a conversion constructor within the C++ proxy class, the conversion constructor for creating a stand-alone proxy instance of the C++ proxy class initialized to not refer to any Java instance.

121. (Original) The method of claim 111, wherein at least one of the proxy support elements provides a semantic usability to the C++ proxy class that closely corresponds to the semantic usability of a Java cast expression corresponding to the Java component.

122. (Original) The method of claim 121, wherein act (b)(i) includes:

generating a static class method within the C++ proxy class, the static class method providing the semantic usability to the C++ proxy class that closely corresponds to the semantic usability of a Java cast expression corresponding to the Java component.

123. (Original) The method of claim 111, wherein at least one of the proxy support elements provides an ability to map an instance of the C++ proxy class to an object that represents the Java component.

124. (Original) The method of claim 123, wherein act (b)(i) includes:

generating a framework support method within the C++ proxy class, the framework support method providing the ability to map an instance of the C++ proxy class to an object that represents the Java component.

125. (Original) The method of claim 91, wherein one or more of the proxy support elements allow an instance of the C++ proxy class to be context-aware.

126. (Original) The method of claim 90, wherein the Java class is declared abstract, and act (b) includes:

defining the C++ proxy component to be instantiable.

127. (Original) The method of claim 93, wherein one or more of the proxy support elements allow an instance of the proxy component to be context-aware.

128. (Original) The method of claim 92 , wherein the Java array has a length attribute, and act (b) includes:

defining the proxy class to include a length field corresponding to the length attribute of the Java array.

129. (Original) The method of claim 92, wherein the Java array has an element type, and act (b) includes:

defining the C++ proxy class to have an element type corresponding to the element type of the Java array; and

generating, within the C++ proxy class, one or more array subscript operators that return a context-aware instance of the proxy class's type.

130. (Original) The method of claim 92, wherein the Java array has a primitive element type, and act (b) includes:

defining the C++ proxy class to have a primitive element type corresponding to the Java array primitive element type; and

generating one or more array subscript operators that return a primitive value.

131. (Original) The method of claim 94, wherein act (b) includes:

defining the C++ proxy method to have a constness based on a mutability of the Java method.

132. (Original) The method of claim 94, wherein act (b) includes:

defining the C++ proxy method to declare a return type that has a mutability based on a mutability of a return type declared by the Java method.

133. (Original) The method of claim 94, wherein act (b) includes:

defining the C++ proxy method to pass one or more arguments; and

defining each argument to have a mutability based on a mutability of a corresponding argument passed by the Java method.

134. (Original) The method of claim 94, wherein act (b) includes:

defining the C++ proxy method to throw an exception based on the Java method being defined to throw an exception.

135. (Original) The method of claim 94, wherein act (b) includes:

defining the C++ proxy method not to throw an exception based on the Java method being defined to throw an exception.

136. (Original) The method of claim 94, wherein act (b) includes:

determining whether the C++ proxy method is declared virtual or not declared virtual from one or more of the following aspects of the Java method: polymorphicability, mutability, and instantiability; and

defining the C++ proxy method to be declared virtual or not declared virtual based on the determination.

137. (Original) The method of claim 95, wherein act (b) includes:

declaring the C++ proxy field to be of type C++ proxy class; and

generating, within the C++ proxy class, one or more proxy support elements that allow an instance of the C++ proxy class to be context-aware, such that an instance of the C++ proxy field is context-aware.

138. (Original) The method of claim 95, wherein the Java field is of primitive type, act (b) including:

defining the C++ proxy field to be declared of type C++ primitive proxy class.

139. (Original) The method of claim 138, wherein act (b) further includes:

declaring the C++ proxy field to be of type C++ proxy class; and

generating, within the C++ proxy class, one or more proxy support elements that allow an instance of the C++ proxy class to be context-aware, such that an instance of the C++ proxy field is context-aware.

140. (Original) The method of claim 138, wherein one or more of the proxy support elements of the C++ primitive proxy class is an overloaded operator that permits instances the C++ primitive proxy class 52 to be used on the left-hand side of one or more syntactical productions.

141. (Original) The method of claim 95, wherein act (b) includes:

determining a mutability of the Java field; and

defining the C++ proxy field to have a mutability determined from the mutability of the Java field.

142. (Original) The method of claim 89, wherein:

act (a) includes determining that the type of the first component is a Java interface, and

act (b) includes transforming the Java interface into a C++ proxy class.

143. (Original) The method of claim 142, wherein act (b) includes:

generating, within the C++ proxy class, one or more proxy support elements.

144. (Original) The method of claim 143, wherein one or more of the proxy support elements allow an instance of the C++ proxy class to be context-aware.

145. (Original) The method of claim 142, wherein act (b) includes:

defining the C++ proxy class to be instantiable.

146. (new) A method comprising,

for a program component expressed in a first domain, the component being of any arbitrary type belonging to a set of different types of program components expressed in the first domain, the type having first relationships to other types in the set,

using a predefined mapping of relationships between types in the set to relationships between types in a second domain to transform the first component to a second component in the second domain, the second component being of a type belonging to a set of different types of program components in the second domain, the type having relationships to components of other types in the second domain that closely correspond to the first relationships.